

## NAG Library Function Document

### nag\_pls\_orth\_scores\_svd (g02lac)

#### 1 Purpose

nag\_pls\_orth\_scores\_svd (g02lac) fits an orthogonal scores partial least squares (PLS) regression by using singular value decomposition.

#### 2 Specification

```
#include <nag.h>
#include <nagg02.h>

void nag_pls_orth_scores_svd (Nag_OrderType order, Integer n, Integer mx,
    const double x[], Integer pdx, const Integer isx[], Integer ip, Integer my,
    const double y[], Integer pdy, double xbar[], double ybar[],
    Nag_ScalePredictor iscale, double xstd[], double ystd[], Integer maxfac,
    double xres[], Integer pdxres, double yres[], Integer pdyres, double w[],
    Integer pdw, double p[], Integer pdp, double t[], Integer pdt, double c[],
    Integer pdc, double u[], Integer pdu, double xcv[], double ycv[],
    Integer pdycv, NagError *fail)
```

#### 3 Description

Let  $X_1$  be the mean-centred  $n$  by  $m$  data matrix  $X$  of  $n$  observations on  $m$  predictor variables. Let  $Y_1$  be the mean-centred  $n$  by  $r$  data matrix  $Y$  of  $n$  observations on  $r$  response variables.

The first of the  $k$  factors PLS methods extract from the data predicts both  $X_1$  and  $Y_1$  by regressing on  $t_1$  a column vector of  $n$  scores:

$$\begin{aligned}\hat{X}_1 &= t_1 p_1^T \\ \hat{Y}_1 &= t_1 c_1^T, \quad \text{with } t_1^T t_1 = 1,\end{aligned}$$

where the column vectors of  $m$   $x$ -loadings  $p_1$  and  $r$   $y$ -loadings  $c_1$  are calculated in the least squares sense:

$$\begin{aligned}p_1^T &= t_1^T X_1 \\ c_1^T &= t_1^T Y_1.\end{aligned}$$

The  $x$ -score vector  $t_1 = X_1 w_1$  is the linear combination of predictor data  $X_1$  that has maximum covariance with the  $y$ -scores  $u_1 = Y_1 c_1$ , where the  $x$ -weights vector  $w_1$  is the normalised first left singular vector of  $X_1^T Y_1$ .

The method extracts subsequent PLS factors by repeating the above process with the residual matrices:

$$\begin{aligned}X_i &= X_{i-1} - \hat{X}_{i-1} \\ Y_i &= Y_{i-1} - \hat{Y}_{i-1}, \quad i = 2, 3, \dots, k,\end{aligned}$$

and with orthogonal scores:

$$t_i^T t_j = 0, \quad j = 1, 2, \dots, i - 1.$$

Optionally, in addition to being mean-centred, the data matrices  $X_1$  and  $Y_1$  may be scaled by standard deviations of the variables. If data are supplied mean-centred, the calculations are not affected within numerical accuracy.

#### 4 References

None.

## 5 Arguments

- 1: **order** – Nag\_OrderType *Input*  
*On entry:* the **order** argument specifies the two-dimensional storage scheme being used, i.e., row-major ordering or column-major ordering. C language defined storage is specified by **order** = Nag\_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.  
*Constraint:* **order** = Nag\_RowMajor or Nag\_ColMajor.
- 2: **n** – Integer *Input*  
*On entry:*  $n$ , the number of observations.  
*Constraint:*  $n > 1$ .
- 3: **mx** – Integer *Input*  
*On entry:* the number of predictor variables.  
*Constraint:*  $mx > 1$ .
- 4: **x**[*dim*] – const double *Input*  
**Note:** the dimension, *dim*, of the array **x** must be at least  
 $\max(1, \mathbf{pdx} \times \mathbf{mx})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{n} \times \mathbf{pdx})$  when **order** = Nag\_RowMajor.  
Where  $\mathbf{X}(i, j)$  appears in this document, it refers to the array element  
 $\mathbf{x}[(j - 1) \times \mathbf{pdx} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{x}[(i - 1) \times \mathbf{pdx} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:*  $\mathbf{X}(i, j)$  must contain the  $i$ th observation on the  $j$ th predictor variable, for  $i = 1, 2, \dots, \mathbf{n}$  and  $j = 1, 2, \dots, \mathbf{mx}$ .
- 5: **pdx** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **x**.  
*Constraints:*  
if **order** = Nag\_ColMajor,  $\mathbf{pdx} \geq \mathbf{n}$ ;  
if **order** = Nag\_RowMajor,  $\mathbf{pdx} \geq \mathbf{mx}$ .
- 6: **isx**[**mx**] – const Integer *Input*  
*On entry:* indicates which predictor variables are to be included in the model.  
 $\mathbf{isx}[j - 1] = 1$   
The  $j$ th predictor variable (with variates in the  $j$ th column of  $X$ ) is included in the model.  
 $\mathbf{isx}[j - 1] = 0$   
Otherwise.  
*Constraint:* the sum of elements in **isx** must equal **ip**.
- 7: **ip** – Integer *Input*  
*On entry:*  $m$ , the number of predictor variables in the model.  
*Constraint:*  $1 < \mathbf{ip} \leq \mathbf{mx}$ .

- 8: **my** – Integer *Input*  
*On entry:*  $r$ , the number of response variables.  
*Constraint:*  $\mathbf{my} \geq 1$ .
- 9: **y[dim]** – const double *Input*  
**Note:** the dimension,  $dim$ , of the array **y** must be at least  
 $\max(1, \mathbf{pdy} \times \mathbf{my})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{n} \times \mathbf{pdy})$  when **order** = Nag\_RowMajor.  
Where  $\mathbf{Y}(i, j)$  appears in this document, it refers to the array element  
 $\mathbf{y}[(j - 1) \times \mathbf{pdy} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{y}[(i - 1) \times \mathbf{pdy} + j - 1]$  when **order** = Nag\_RowMajor.  
*On entry:*  $\mathbf{Y}(i, j)$  must contain the  $i$ th observation for the  $j$ th response variable, for  $i = 1, 2, \dots, \mathbf{n}$   
and  $j = 1, 2, \dots, \mathbf{my}$ .
- 10: **pdy** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **y**.  
*Constraints:*  
if **order** = Nag\_ColMajor,  $\mathbf{pdy} \geq \mathbf{n}$ ;  
if **order** = Nag\_RowMajor,  $\mathbf{pdy} \geq \mathbf{my}$ .
- 11: **xbar[ip]** – double *Output*  
*On exit:* mean values of predictor variables in the model.
- 12: **ybar[my]** – double *Output*  
*On exit:* the mean value of each response variable.
- 13: **iscale** – Nag\_ScalePredictor *Input*  
*On entry:* indicates how predictor variables are scaled.  
**iscale** = Nag\_PredStdScale  
Data are scaled by the standard deviation of variables.  
**iscale** = Nag\_PredUserScale  
Data are scaled by user-supplied scalings.  
**iscale** = Nag\_PredNoScale  
No scaling.  
*Constraint:* **iscale** = Nag\_PredNoScale, Nag\_PredStdScale or Nag\_PredUserScale.
- 14: **xstd[ip]** – double *Input/Output*  
*On entry:* if **iscale** = Nag\_PredUserScale, **xstd**[ $j - 1$ ] must contain the user-supplied scaling for the  $j$ th predictor variable in the model, for  $j = 1, 2, \dots, \mathbf{ip}$ . Otherwise **xstd** need not be set.  
*On exit:* if **iscale** = Nag\_PredStdScale, standard deviations of predictor variables in the model. Otherwise **xstd** is not changed.
- 15: **ystd[my]** – double *Input/Output*  
*On entry:* if **iscale** = Nag\_PredUserScale, **ystd**[ $j - 1$ ] must contain the user-supplied scaling for the  $j$ th response variable in the model, for  $j = 1, 2, \dots, \mathbf{my}$ . Otherwise **ystd** need not be set.  
*On exit:* if **iscale** = Nag\_PredStdScale, the standard deviation of each response variable. Otherwise **ystd** is not changed.

- 16: **maxfac** – Integer *Input*  
*On entry:*  $k$ , the number of latent variables to calculate.  
*Constraint:*  $1 \leq \mathbf{maxfac} \leq \mathbf{ip}$ .
- 17: **xres**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **xres** must be at least  
 $\max(1, \mathbf{pdxres} \times \mathbf{ip})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{n} \times \mathbf{pdxres})$  when **order** = Nag\_RowMajor.  
The ( $i, j$ )th element of the matrix is stored in  
 $\mathbf{xres}[(j - 1) \times \mathbf{pdxres} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{xres}[(i - 1) \times \mathbf{pdxres} + j - 1]$  when **order** = Nag\_RowMajor.  
*On exit:* the predictor variables' residual matrix  $X_k$ .
- 18: **pdxres** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **xres**.  
*Constraints:*  
if **order** = Nag\_ColMajor, **pdxres**  $\geq \mathbf{n}$ ;  
if **order** = Nag\_RowMajor, **pdxres**  $\geq \mathbf{ip}$ .
- 19: **yres**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **yres** must be at least  
 $\max(1, \mathbf{pdyres} \times \mathbf{my})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{n} \times \mathbf{pdyres})$  when **order** = Nag\_RowMajor.  
The ( $i, j$ )th element of the matrix is stored in  
 $\mathbf{yres}[(j - 1) \times \mathbf{pdyres} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{yres}[(i - 1) \times \mathbf{pdyres} + j - 1]$  when **order** = Nag\_RowMajor.  
*On exit:* the residuals for each response variable,  $Y_k$ .
- 20: **pdyres** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **yres**.  
*Constraints:*  
if **order** = Nag\_ColMajor, **pdyres**  $\geq \mathbf{n}$ ;  
if **order** = Nag\_RowMajor, **pdyres**  $\geq \mathbf{my}$ .
- 21: **w**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **w** must be at least  
 $\max(1, \mathbf{pdw} \times \mathbf{maxfac})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{ip} \times \mathbf{pdw})$  when **order** = Nag\_RowMajor.  
The ( $i, j$ )th element of the matrix  $W$  is stored in  
 $\mathbf{w}[(j - 1) \times \mathbf{pdw} + i - 1]$  when **order** = Nag\_ColMajor;  
 $\mathbf{w}[(i - 1) \times \mathbf{pdw} + j - 1]$  when **order** = Nag\_RowMajor.  
*On exit:* the  $j$ th column of  $W$  contains the  $x$ -weights  $w_j$ , for  $j = 1, 2, \dots, \mathbf{maxfac}$ .

- 22: **pdw** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **w**.  
*Constraints:*  
 if **order** = Nag\_ColMajor, **pdw** ≥ **ip**;  
 if **order** = Nag\_RowMajor, **pdw** ≥ **maxfac**.
- 23: **p**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **p** must be at least  
 max(1, **pdp** × **maxfac**) when **order** = Nag\_ColMajor;  
 max(1, **ip** × **pdp**) when **order** = Nag\_RowMajor.  
 The (*i*, *j*)th element of the matrix *P* is stored in  
**p**[(*j* – 1) × **pdp** + *i* – 1] when **order** = Nag\_ColMajor;  
**p**[(*i* – 1) × **pdp** + *j* – 1] when **order** = Nag\_RowMajor.  
*On exit:* the *j*th column of *P* contains the *x*-loadings *p<sub>j</sub>*, for *j* = 1, 2, . . . , **maxfac**.
- 24: **pdp** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **p**.  
*Constraints:*  
 if **order** = Nag\_ColMajor, **pdp** ≥ **ip**;  
 if **order** = Nag\_RowMajor, **pdp** ≥ **maxfac**.
- 25: **t**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **t** must be at least  
 max(1, **pdt** × **maxfac**) when **order** = Nag\_ColMajor;  
 max(1, **n** × **pdt**) when **order** = Nag\_RowMajor.  
 The (*i*, *j*)th element of the matrix *T* is stored in  
**t**[(*j* – 1) × **pdt** + *i* – 1] when **order** = Nag\_ColMajor;  
**t**[(*i* – 1) × **pdt** + *j* – 1] when **order** = Nag\_RowMajor.  
*On exit:* the *j*th column of *T* contains the *x*-scores *t<sub>j</sub>*, for *j* = 1, 2, . . . , **maxfac**.
- 26: **pdt** – Integer *Input*  
*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **t**.  
*Constraints:*  
 if **order** = Nag\_ColMajor, **pdt** ≥ **n**;  
 if **order** = Nag\_RowMajor, **pdt** ≥ **maxfac**.
- 27: **c**[*dim*] – double *Output*  
**Note:** the dimension, *dim*, of the array **c** must be at least  
 max(1, **cdc** × **maxfac**) when **order** = Nag\_ColMajor;  
 max(1, **my** × **cdc**) when **order** = Nag\_RowMajor.  
 The (*i*, *j*)th element of the matrix *C* is stored in  
**c**[(*j* – 1) × **cdc** + *i* – 1] when **order** = Nag\_ColMajor;  
**c**[(*i* – 1) × **cdc** + *j* – 1] when **order** = Nag\_RowMajor.

*On exit:* the  $j$ th column of  $C$  contains the  $y$ -loadings  $c_j$ , for  $j = 1, 2, \dots, \mathbf{maxfac}$ .

28: **pdc** – Integer *Input*

*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **c**.

*Constraints:*

if **order** = Nag\_ColMajor, **pdc**  $\geq$  **my**;  
if **order** = Nag\_RowMajor, **pdc**  $\geq$  **maxfac**.

29: **u**[*dim*] – double *Output*

**Note:** the dimension, *dim*, of the array **u** must be at least

$\max(1, \mathbf{pdu} \times \mathbf{maxfac})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{n} \times \mathbf{pdu})$  when **order** = Nag\_RowMajor.

The ( $i, j$ )th element of the matrix  $U$  is stored in

**u**[( $j - 1$ )  $\times$  **pdu** +  $i - 1$ ] when **order** = Nag\_ColMajor;  
**u**[( $i - 1$ )  $\times$  **pdu** +  $j - 1$ ] when **order** = Nag\_RowMajor.

*On exit:* the  $j$ th column of  $U$  contains the  $y$ -scores  $u_j$ , for  $j = 1, 2, \dots, \mathbf{maxfac}$ .

30: **pdu** – Integer *Input*

*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **u**.

*Constraints:*

if **order** = Nag\_ColMajor, **pdu**  $\geq$  **n**;  
if **order** = Nag\_RowMajor, **pdu**  $\geq$  **maxfac**.

31: **xcv**[**maxfac**] – double *Output*

*On exit:* **xcv**[ $j - 1$ ] contains the cumulative percentage of variance in the predictor variables explained by the first  $j$  factors, for  $j = 1, 2, \dots, \mathbf{maxfac}$ .

32: **ycv**[*dim*] – double *Output*

**Note:** the dimension, *dim*, of the array **ycv** must be at least

$\max(1, \mathbf{pdycv} \times \mathbf{my})$  when **order** = Nag\_ColMajor;  
 $\max(1, \mathbf{maxfac} \times \mathbf{pdycv})$  when **order** = Nag\_RowMajor.

Where **YCV**( $i, j$ ) appears in this document, it refers to the array element

**ycv**[( $j - 1$ )  $\times$  **pdycv** +  $i - 1$ ] when **order** = Nag\_ColMajor;  
**ycv**[( $i - 1$ )  $\times$  **pdycv** +  $j - 1$ ] when **order** = Nag\_RowMajor.

*On exit:* **YCV**( $i, j$ ) is the cumulative percentage of variance of the  $j$ th response variable explained by the first  $i$  factors, for  $i = 1, 2, \dots, \mathbf{maxfac}$  and  $j = 1, 2, \dots, \mathbf{my}$ .

33: **pdycv** – Integer *Input*

*On entry:* the stride separating matrix row or column elements (depending on the value of **order**) in **ycv**.

*Constraints:*

if **order** = Nag\_ColMajor, **pdycv**  $\geq$  **maxfac**;  
if **order** = Nag\_RowMajor, **pdycv**  $\geq$  **my**.

34: **fail** – NagError \*

Input/Output

The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **mx** =  $\langle value \rangle$ .

Constraint: **mx** > 1.

On entry, **my** =  $\langle value \rangle$ .

Constraint: **my** ≥ 1.

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n** > 1.

On entry, **pdc** =  $\langle value \rangle$ .

Constraint: **pdc** > 0.

On entry, **pdp** =  $\langle value \rangle$ .

Constraint: **pdp** > 0.

On entry, **pdt** =  $\langle value \rangle$ .

Constraint: **pdt** > 0.

On entry, **pdu** =  $\langle value \rangle$ .

Constraint: **pdu** > 0.

On entry, **pdw** =  $\langle value \rangle$ .

Constraint: **pdw** > 0.

On entry, **pdx** =  $\langle value \rangle$ .

Constraint: **pdx** > 0.

On entry, **pdxres** =  $\langle value \rangle$ .

Constraint: **pdxres** > 0.

On entry, **pdy** =  $\langle value \rangle$ .

Constraint: **pdy** > 0.

On entry, **pdycv** =  $\langle value \rangle$ .

Constraint: **pdycv** > 0.

On entry, **pdyres** =  $\langle value \rangle$ .

Constraint: **pdyres** > 0.

### NE\_INT\_2

On entry, **ip** =  $\langle value \rangle$ , **mx** =  $\langle value \rangle$ .

Constraint:  $1 < \mathbf{ip} \leq \mathbf{mx}$ .

On entry, **maxfac** =  $\langle value \rangle$ , **ip** =  $\langle value \rangle$ .

Constraint:  $1 \leq \mathbf{maxfac} \leq \mathbf{ip}$ .

On entry, **pdc** =  $\langle value \rangle$ , **maxfac** =  $\langle value \rangle$ .

Constraint: **pdc** ≥ **maxfac**.

On entry, **pdc** =  $\langle value \rangle$ , **my** =  $\langle value \rangle$ .

Constraint: **pdc** ≥ **my**.

On entry, **pdp** =  $\langle value \rangle$ , **ip** =  $\langle value \rangle$ .

Constraint: **pdp**  $\geq$  **ip**.

On entry, **pdp** =  $\langle value \rangle$ , **maxfac** =  $\langle value \rangle$ .

Constraint: **pdp**  $\geq$  **maxfac**.

On entry, **pdt** =  $\langle value \rangle$ , **maxfac** =  $\langle value \rangle$ .

Constraint: **pdt**  $\geq$  **maxfac**.

On entry, **pdt** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .

Constraint: **pdt**  $\geq$  **n**.

On entry, **pdu** =  $\langle value \rangle$ , **maxfac** =  $\langle value \rangle$ .

Constraint: **pdu**  $\geq$  **maxfac**.

On entry, **pdu** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .

Constraint: **pdu**  $\geq$  **n**.

On entry, **pdw** =  $\langle value \rangle$ , **ip** =  $\langle value \rangle$ .

Constraint: **pdw**  $\geq$  **ip**.

On entry, **pdw** =  $\langle value \rangle$ , **maxfac** =  $\langle value \rangle$ .

Constraint: **pdw**  $\geq$  **maxfac**.

On entry, **pdx** =  $\langle value \rangle$ , **mx** =  $\langle value \rangle$ .

Constraint: **pdx**  $\geq$  **mx**.

On entry, **pdx** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .

Constraint: **pdx**  $\geq$  **n**.

On entry, **pdxres** =  $\langle value \rangle$ , **ip** =  $\langle value \rangle$ .

Constraint: **pdxres**  $\geq$  **ip**.

On entry, **pdxres** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .

Constraint: **pdxres**  $\geq$  **n**.

On entry, **pdycv** =  $\langle value \rangle$ , **my** =  $\langle value \rangle$ .

Constraint: **pdycv**  $\geq$  **my**.

On entry, **pdycv** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .

Constraint: **pdycv**  $\geq$  **n**.

On entry, **pdycv** =  $\langle value \rangle$ , **maxfac** =  $\langle value \rangle$ .

Constraint: **pdycv**  $\geq$  **maxfac**.

On entry, **pdycv** =  $\langle value \rangle$ , **my** =  $\langle value \rangle$ .

Constraint: **pdycv**  $\geq$  **my**.

On entry, **pdycv** =  $\langle value \rangle$ , **my** =  $\langle value \rangle$ .

Constraint: **pdycv**  $\geq$  **my**.

On entry, **pdycv** =  $\langle value \rangle$ , **n** =  $\langle value \rangle$ .

Constraint: **pdycv**  $\geq$  **n**.

#### NE\_INT\_ARG\_CONS

On entry, **ip** is not equal to the sum of **isx** elements: **ip** =  $\langle value \rangle$ ,  $\text{sum}(\mathbf{isx}) = \langle value \rangle$ .

#### NE\_INT\_ARRAY\_VAL\_1\_OR\_2

On entry, element  $\langle value \rangle$  of **isx** is invalid.

#### NE\_INTERNAL\_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.



## 7 Accuracy

The computed singular value decomposition is nearly the exact singular value decomposition for a nearby matrix  $(A + E)$ , where

$$\|E\|_2 = O(\epsilon)\|A\|_2,$$

and  $\epsilon$  is the *machine precision*.

## 8 Further Comments

`nag_pls_orth_scores_svd` (g02lac) allocates internally  $2mr + A + \max(3(A + B), 5A) + r$  elements of double storage, where  $A = \min(m, r)$  and  $B = \max(m, r)$ .

## 9 Example

This example reads in data from an experiment to measure the biological activity in a chemical compound, and a PLS model is estimated.

### 9.1 Program Text

```

/* nag_pls_orth_scores_svd (g02lac) Example Program.
 *
 * Copyright 2008, Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg02.h>
#include <nagx04.h>

int main(int argc, char *argv[])
{
    FILE          *fpin, *fpout;
    char          *outfile = 0;
    /*Integer scalar and array declarations */
    Integer       exit_status = 0;
    Integer       i, ip, j, maxfac, mx, my, n;
    Integer       pdc, pdp, pdt, pdu, pdw, pdx, pdxres, pdy, pdycv, pdyres;
    Integer       *isx = 0;
    /*Double scalar and array declarations */
    double        *c = 0, *p = 0, *t = 0, *u = 0, *w = 0, *x = 0, *xbar = 0;
    double        *xcv = 0, *xres = 0, *xstd = 0, *y = 0, *ybar = 0;
    double        *ycv = 0, *yres = 0, *ystd = 0;
    /*Character scalar and array declarations */
    char          sscale[18];
    /*NAG Types */
    Nag_OrderType order;
    Nag_ScalePredictor scale;
    NagError      fail;

    INIT_FAIL(fail);

    /* Check for command-line IO options */
    fpin = nag_example_file_io(argc, argv, "-data", NULL);
    fpout = nag_example_file_io(argc, argv, "-results", NULL);
    (void) nag_example_file_io(argc, argv, "-nag_write", &outfile);

    fprintf(fpout,
            "nag_pls_orth_scores_svd (g02lac) Example Program Results\n");
    /* Skip header in data file.*/
    fscanf(fpin, "%*[\n] ");
    /* Read data values.*/

```

```

fscanf(fpin, "%ld%ld%ld%s %ld%*[\n] ",
      &n, &mx, &my, sscale, &maxfac);
scale = (Nag_ScalePredictor) nag_enum_name_to_value(sscale);

if (!(isx = NAG_ALLOC(mx, Integer)))
{
  fprintf(fpout, "Allocation failure\n");
  exit_status = -1;
  goto END;
}
for (j = 0; j < mx; j++)
  fscanf(fpin, "%ld ", &isx[j]);
fscanf(fpin, "%*[\n] ");
ip = 0;
for (j = 0; j < mx; j++)
{
  if (isx[j] == 1)
    ip = ip+1;
}
#ifdef NAG_COLUMN_MAJOR
pdc = my;
#define C(I, J)    c[(J-1)*pdc + I-1]
pdp = ip;
#define P(I, J)    p[(J-1)*pdp + I-1]
pdt = n;
#define T(I, J)    t[(J-1)*pdt + I-1]
pdu = n;
#define U(I, J)    u[(J-1)*pdu + I-1]
pdw = ip;
#define W(I, J)    w[(J-1)*pdw + I-1]
pdx = n;
#define X(I, J)    x[(J-1)*pdx + I-1]
pdxres = n;
#define XRES(I, J) xres[(J-1)*pdxres + I-1]
pdy = n;
#define Y(I, J)    y[(J-1)*pdy + I-1]
pdycv = maxfac;
#define YCV(I, J)  ycv[(J-1)*pdycv + I-1]
pdyres = n;
#define YRES(I, J) yres[(J-1)*pdyres + I-1]
order = Nag_ColMajor;
#else
pdc = maxfac;
#define C(I, J)    c[(I-1)*pdc + J-1]
pdp = maxfac;
#define P(I, J)    p[(I-1)*pdp + J-1]
pdt = maxfac;
#define T(I, J)    t[(I-1)*pdt + J-1]
pdu = maxfac;
#define U(I, J)    u[(I-1)*pdu + J-1]
pdw = maxfac;
#define W(I, J)    w[(I-1)*pdw + J-1]
pdx = mx;
#define X(I, J)    x[(I-1)*pdx + J-1]
pdxres = ip;
#define XRES(I, J) xres[(I-1)*pdxres + J-1]
pdy = my;
#define Y(I, J)    y[(I-1)*pdy + J-1]
pdycv = my;
#define YCV(I, J)  ycv[(I-1)*pdycv + J-1]
pdyres = my;
#define YRES(I, J) yres[(I-1)*pdyres + J-1]
order = Nag_RowMajor;
#endif
/* Assign parameter values to corresponding variables */
if (!(c = NAG_ALLOC(pdc*(order == Nag_RowMajor?my:maxfac), double)) ||
    !(p = NAG_ALLOC(pdp*(order == Nag_RowMajor?ip:maxfac), double)) ||
    !(t = NAG_ALLOC(pdt*(order == Nag_RowMajor?n:maxfac), double)) ||
    !(u = NAG_ALLOC(pdu*(order == Nag_RowMajor?n:maxfac), double)) ||
    !(w = NAG_ALLOC(pdw*(order == Nag_RowMajor?ip:maxfac), double)) ||
    !(x = NAG_ALLOC(pdx*(order == Nag_RowMajor?n:mx), double)) ||

```

```

!(xbar = NAG_ALLOC(ip, double)) ||
!(xcv = NAG_ALLOC(maxfac, double)) ||
!(xres = NAG_ALLOC(pdxres*(order == Nag_RowMajor?n:ip), double)) ||
!(xstd = NAG_ALLOC(ip, double)) ||
!(y = NAG_ALLOC(pdy*(order == Nag_RowMajor?n:my), double)) ||
!(ybar = NAG_ALLOC(my, double)) ||
!(ycv = NAG_ALLOC(pdycv*(order == Nag_RowMajor?maxfac:my),
double)) ||
!(yres = NAG_ALLOC(pdyres*(order == Nag_RowMajor?n:my), double)) ||
!(ystd = NAG_ALLOC(my, double))
{
fprintf(fpout, "Allocation failure\n");
exit_status = -1;
goto END;
}
/* Read data values.*/
for (i = 1; i <= n; i++)
{
for (j = 1; j <= mx; j++)
fscanf(fpin, "%lf ", &X(i, j));
for (j = 1; j <= my; j++)
fscanf(fpin, "%lf ", &Y(i, j));
}
fscanf(fpin, "%*[\n] ");
/* Fit a PLS model.*/
/*
* nag_pls_orth_scores_svd (g02lac)
* Partial least-squares
*/
nag_pls_orth_scores_svd(order, n, mx, x, pdx, isx, ip, my, y, pdy, xbar,
ybar, scale, xstd, ystd, maxfac, xres, pdxres,
yres, pdyres, w, pdw, p, pdp, t, pdt, c, pdc, u,
pdu, xcv, ycv, pdycv, &fail);
if (fail.code != NE_NOERROR)
{
fprintf(fpout, "Error from nag_pls_orth_scores_svd (g02lac).\n%s\n",
fail.message);
exit_status = 1;
goto END;
}
/*
* nag_gen_real_mat_print (x04cac)
* Print real general matrix (easy-to-use)
*/
if (outfile) fclose(fpout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, ip,
maxfac, p, pdp, "x-loadings, P", outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
exit_status = 2;
goto END;
}
if (fail.code != NE_NOERROR)
{
fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
fail.message);
exit_status = 1;
goto END;
}
/*
* nag_gen_real_mat_print (x04cac)
* Print real general matrix (easy-to-use)
*/
if (outfile) fclose(fpout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
maxfac, t, pdt, "x-scores, T", outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
exit_status = 2;
goto END;
}
}

```

```

if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}
/*
 * nag_gen_real_mat_print (x04cac)
 * Print real general matrix (easy-to-use)
 */
if (outfile) fclose(fpout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, my,
                      maxfac, c, pdc, "y-loadings, C", outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}
/*
 * nag_gen_real_mat_print (x04cac)
 * Print real general matrix (easy-to-use)
 */
if (outfile) fclose(fpout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
                      maxfac, u, pdu, "y-scores, U", outfile, &fail);
if (outfile && !(fpout = fopen(outfile, "a")))
{
    exit_status = 2;
    goto END;
}
if (fail.code != NE_NOERROR)
{
    fprintf(fpout, "Error from nag_gen_real_mat_print (x04cac).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
}
}
fprintf(fpout, "\n");
fprintf(fpout, "%s\n", "Explained Variance");
fprintf(fpout, "%12s%24s\n", "Model effects", "Dependent variable(s)");
for (i = 1; i <= maxfac; i++)
{
    fprintf(fpout, "%12.6f", xcv[i-1]);
    for (j = 1; j <= my; j++)
        fprintf(fpout, "%12.6f%s", YCV(i, j), j%10?" ":"\n");
    fprintf(fpout, "\n");
}
}
END:
if (fpin != stdin) fclose(fpin);
if (fpout != stdout) fclose(fpout);
if (c) NAG_FREE(c);
if (p) NAG_FREE(p);
if (t) NAG_FREE(t);
if (u) NAG_FREE(u);
if (w) NAG_FREE(w);
if (x) NAG_FREE(x);
if (xbar) NAG_FREE(xbar);
if (xcv) NAG_FREE(xcv);
if (xres) NAG_FREE(xres);
if (xstd) NAG_FREE(xstd);
if (y) NAG_FREE(y);
if (ybar) NAG_FREE(ybar);

```

```

if (ycv) NAG_FREE(ycv);
if (yres) NAG_FREE(yres);
if (ystd) NAG_FREE(ystd);
if (isx) NAG_FREE(isx);

return exit_status;
}

```

## 9.2 Program Data

```

nag_pls_orth_scores_svd (g02lac) Example Program Data
15 15 1 Nag_PredStdScale 4 : n, mx, my, iscale, maxfac
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 : isx
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 1.9607 -1.6324 0.5746
1.9607 -1.6324 0.574 2.8369 1.4092 -3.1398 0.00
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 1.9607 -1.6324 0.5746
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 0.28
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 0.0744 -1.7333 0.0902
1.9607 -1.6324 0.5746 2.8369 1.4092 -3.1398 0.20
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 0.0744 -1.7333 0.0902
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 0.51
-2.6931 -2.5271 -1.2871 2.8369 1.4092 -3.1398 0.0744 -1.7333 0.0902
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 0.11
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 -4.7548 3.6521 0.8524
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 2.73
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 0.0744 -1.7333 0.0902
0.0744 -1.7333 0.0902 -1.2201 0.8829 2.2253 0.18
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 2.4064 1.7438 1.1057
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 1.53
-2.6931 -2.5271 -1.2871 0.0744 -1.7333 0.0902 0.0744 -1.7333 0.0902
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 -0.10
2.2261 -5.3648 0.3049 3.0777 0.3891 -0.0701 0.0744 -1.7333 0.0902
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 -0.52
-4.1921 -1.0285 -0.9801 3.0777 0.3891 -0.0701 0.0744 -1.7333 0.0902
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 0.40
-4.9217 1.2977 0.4473 3.0777 0.3891 -0.0701 0.0744 -1.7333 0.0902
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 0.30
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 2.2261 -5.3648 0.3049
2.2261 -5.3648 0.3049 2.8369 1.4092 -3.1398 -1.00
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 -4.9217 1.2977 0.4473
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 1.57
-2.6931 -2.5271 -1.2871 3.0777 0.3891 -0.0701 -4.1921 -1.0285 -0.9801
0.0744 -1.7333 0.0902 2.8369 1.4092 -3.1398 0.59 : End of observations

```

## 9.3 Program Results

```

nag_pls_orth_scores_svd (g02lac) Example Program Results
x-loadings, P

```

	1	2	3	4
1	-0.6708	-1.0047	0.6505	0.6169
2	0.4943	0.1355	-0.9010	-0.2388
3	-0.4167	-1.9983	-0.5538	0.8474
4	0.3930	1.2441	-0.6967	-0.4336
5	0.3267	0.5838	-1.4088	-0.6323
6	0.0145	0.9607	1.6594	0.5361
7	-2.4471	0.3532	-1.1321	-1.3554
8	3.5198	0.6005	0.2191	0.0380
9	1.0973	2.0635	-0.4074	-0.3522
10	-2.4466	2.5640	-0.4806	0.3819
11	2.2732	-1.3110	-0.7686	-1.8959
12	-1.7987	2.4088	-0.9475	-0.4727
13	0.3629	0.2241	-2.6332	2.3739
14	0.3629	0.2241	-2.6332	2.3739
15	-0.3629	-0.2241	2.6332	-2.3739

```

x-scores, T

```

	1	2	3	4
1	-0.1896	0.3898	-0.2502	-0.2479
2	0.0201	-0.0013	-0.1726	-0.2042

3	-0.1889	0.3141	-0.1727	-0.1350
4	0.0210	-0.0773	-0.0950	-0.0912
5	-0.0090	-0.2649	-0.4195	-0.1327
6	0.5479	0.2843	0.1914	0.2727
7	-0.0937	-0.0579	0.6799	-0.6129
8	0.2500	0.2033	-0.1046	-0.1014
9	-0.1005	-0.2992	0.2131	0.1223
10	-0.1810	-0.4427	0.0559	0.2114
11	0.0497	-0.0762	-0.1526	-0.0771
12	0.0173	-0.2517	-0.2104	0.1044
13	-0.6002	0.3596	0.1876	0.4812
14	0.3796	0.1338	0.1410	0.1999
15	0.0773	-0.2139	0.1085	0.2106

y-loadings, C

	1	2	3	4
1	3.5425	1.0475	0.2548	0.1866

y-scores, U

	1	2	3	4
1	-1.7670	0.1812	-0.0600	-0.0320
2	-0.6724	-0.2735	-0.0662	-0.0402
3	-0.9852	0.4097	0.0158	0.0198
4	0.2267	-0.0107	0.0180	0.0177
5	-1.3370	-0.3619	-0.0173	0.0073
6	8.9056	0.6000	0.0701	0.0422
7	-1.0634	0.0332	0.0235	-0.0151
8	4.2143	0.3184	0.0232	0.0219
9	-2.1580	-0.2652	0.0153	0.0011
10	-3.7999	-0.4520	0.0082	0.0034
11	-0.2033	-0.2446	-0.0392	-0.0214
12	-0.5942	-0.2398	0.0089	0.0165
13	-5.6764	0.5487	0.0375	0.0185
14	4.3707	-0.1161	-0.0639	-0.0535
15	0.5395	-0.1274	0.0261	0.0139

## Explained Variance

Model effects	Dependent variable(s)
16.902124	89.638060
29.674338	97.476270
44.332404	97.939839
56.172041	98.188474

---